On Designing an ACL2-Based C Integer Type Safety Checking Tool

Kevin Krause and Jim Alves-Foss,
Center for Secure and Dependable Systems, University of Idaho

Consider the Truth Value

- **`if(x > -10) { // do something important }`**
  - C source code
  - `x` represents a positive integer between 1 and 10 inclusive
- **Truth is Dependant on the Integer Type of `x`**
  - If `x` is an **`unsigned int`** type, then "`do something important`" **would not be executed**
  - If `x` is an **`unsigned`** **integer type** with a **smaller precision than** that of an **`unsigned int`**,, then "`do something important`" **would be executed**

Integer Error Conditions

- **Overflow/Underflow**
  - Occurs whenever the value of an integer type is increased or decreased beyond the type's valid value range
    - `unsigned` integers silently wrap
    - behavior for `signed` integers is undefined
- **Sign error**
  - Occurs whenever the meaning of the high order bit is lost
    - A precision bit
    - A sign bit
- **Truncation Error**
  - Occurs during the coercion from a larger type to a smaller type and the high order bits are truncated
    - `11111111` → `1111`

Static C Typing Semantics

- **Syntax of Types**
  - < *c_type* > := < *object-type* > | < *function-type* > | < *incomplete-type* >
    - < *scalar-type* > := < *arithmetic-type* > | < *pointer-type* >
  - Shows subtyping relationships necessary for C coercion rules
- **Typing Inference Rules Based on Operator/Operand Constraints**
  - Typing judgments of the form $\Gamma \vdash E : \theta$
    - Where E is an expression, Γ is the type environment, and θ is the type attributed to E
  - For example, the multiplicative division operation

$$\frac{\Gamma \vdash e_1 : exp[\tau_1] \quad \Gamma \vdash e_2 : exp[\tau_2] \quad isArithmetic(\tau_1) \quad isArithmetic(\tau_2) \quad \tau' ::= arithConv(\tau_1, \tau_2)}{\Gamma \vdash e_1 \,/\, e_2 : exp[\tau']}$$

Example Output

- ```
  char c1;
  int i1 = 64;
  c1 = i1 * 2;
  ```

- ```
  ((EXPSTMT (ASSN (ID "c1" 2)
                  (MULT (ID "i1" 3)(LIT 2)))
            (LINE 3))
   (2 ("c1")((CHAR)(NOQUAL)(NOSTORE)
      (128) ("Error: exceeds value range of type"
             CHAR))))
  ```

C is Weakly Typed and is Not Type Safe

- **Type Strength**
  - A language characteristic based on the amount of coercion (casting) permitted and performed among its data types
    - Less coercions = stronger typing
  - Coercion is generally performed during compile time to insure compatibility of operator and operand types
  - C integer coercions are rule based
    - Integer ranking
    - Integer promotion rules
    - Usual arithmetic conversions
  - C does not support valid range checking during the coercion process
- **Type Safety**
  - A program property of being free from unexpected results
    - Unexpected results = compromised system state
    - Compromised system state = vulnerable to attacks and/or failure
      - Denial of Service
      - Execution of Arbitrary Code
      - Escalation of Privileges

Project Tasks

- **Formalize C's Static Typing Semantics**
- **Construct Tool Around Formal Static Typing Semantics**
- **Prove Assumptions Made About Both are Correct**

ACL2

- **A Computational Logic**
  - First order theorem prover
- **Applicative Common Lisp**
  - Non-destructive programming language

Tool Functionality

- **Input AST and .symtab from c2acl2 Translator**
- **First Pass:**
  - Extract, model, and model declarations
    - `(SYM ("NAME")((TYPE)(QUAL)(STORE))(VALUE))`
      - updatable lookup table
- **Second and Subsequent Passes:**
  - Analyze Expressions and Statements
    - Check operator/operand compatibility
      - If error found, issue and append error statement
    - Check promoted operand values
      - If error found, issue and append error statement
    - If an assignment expression
      - Evaluate RHS first and LHS second
      - If new LHS value can be determined, validate value and add to lookup table
      - If value cannot be determined, issue a conditional warning
      - If value is in error, append error statement which remains until next assignment statement.
- **Proof Generation**

University of Idaho
College of Engineering